Introducció al simulador de xarxes OMNeT++

Anàlisi i Dimensionat de Xarxes

Grau en Enginyeria Telemàtica

Anna Agustí
José Ramón Piney

Departament d'Enginyeria Telemàtica
Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
Universitat Politècnica de Catalunya

CONTINGUTS

1 INTRODUCCIÓ

OMNeT++ és un entorn i una llibreria de simulació, modular, extensible, i basat en components C++ que proporciona l'esquelet per al desenvolupament de simuladors de xarxa d'esdeveniments discrets. L'arquitectura genèrica d'OMNeT++ en permet l'aplicació en diversos entorns com, per exemple, el modelat de xarxes amb i sense fils, l'anàlisi de protocols, la caracterització de xarxes de cues i, en general, el modelat de qualsevol sistema susceptible de ser caracteritzat mitjançant entitats que es comuniquen intercanviant missatges. OMNeT++ ofereix un entorn de desenvolupament integrat (IDE) basat en l'Eclipse i l'amplia amb nous editors, vistes, *wizards* i d'altres funcionalitats. OMNeT++ aporta les funcionalitats per crear i configurar models, realitzar execucions en bloc i analitzar els resultats de les simulacions, mentre que l'Eclipse proporciona, entre d'altres coses, un entorn amigable de programació en C++.

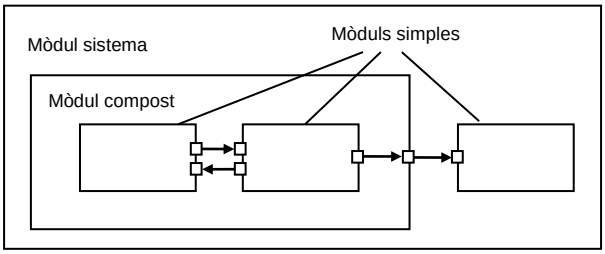
L'entorn de simulació, les interfícies d'usuari i les eines relacionades amb la configuració i l'anàlisi dels resultats són molt portables. S'han testejat en els sistemes operatius més comuns (Linux, Mac OS/X, Windows) i es poden compilar realitzant lleugeres modificacions en la majoria de sistemes operatius basats en Unix.

El simulador és d'ús lliure en l'entorn acadèmic i amb finalitats no lucratives. Per a ús comercial cal obtenir la llicència OMNEST (versió comercial de l'OMNeT++) de Simulcraft Inc.

L'OMNeT++ i tota la documentació relacionada amb la seva instal·lació i funcionament està disponible a la pàgina: http://omnetpp.org/

2 ESTRUCTURA GENERAL

Un model OMNeT++ consisteix en la interconnexió de mòduls que es comuniquen intercanviant missatges. Els **mòduls simples** són els components bàsics d'OMNeT++, estan escrits en C++ i fan servir la llibreria de simulació. La unió de diferents mòduls simples permet la definició de **mòduls compostos**, que, a la vegada, es poden utilitzar en la definició d'altres mòduls compostos definint una estructura jeràrquica que no té límit en el nombre de nivells. El model sencer, s'anomena **xarxa** i es pot considerar, en si mateix, un mòdul compost.



Els mòduls es comuniquen intercanviant missatges definits en C++ que contenen dades arbitràries. Els mòduls, típicament, envien i reben els missatges a través d'accessos (**gates**) que actuen com a interfícies d'entrada i de sortida i que s'interconnecten mitjançant connexions (**connections**). Degut a l'estructura jeràrquica del model, els missatges típicament travessen una cadena de connexions, començant i acabant en mòduls simples. Les connexions poden tenir paràmetres associats com, per exemple, el retard de propagació, la capacitat del canal o la taxa d'error. Anàlogament, els mòduls també poden tenir paràmetres associats que s'utilitzen per configurar els mòduls simples o per definir la topologia del model.

Per poder definir la topologia de la xarxa, els mòduls que la composen, les interconnexions entre ells i els paràmetres relacionats, OMNeT++ utilitza un llenguatge d'alt nivell anomenat NED (*NEtwork Description*). Així doncs, les xarxes (que constitueixen els mòduls de simulació auto-continguts), els mòduls composts i els mòduls simples s'especifiquen segons el format definit pel llenguatge NED i es guarden en fitxers amb extensió *.ned*.

Els models de simulació en OMNeT++ es parametritzen i es configuren per a ser executats utilitzant fitxers de configuració amb extensió *.ini*, anomenats fitxers INI. En aquests fitxers de configuració es poden especificar, per exemple, els valors concrets que es volen utilitzar com a paràmetres d'entrada d'una determinada xarxa, la configuració dels generadors de números aleatoris o el temps límit de simulació.

3 FITXERS NED

El llenguatge NED permet definir la topologia de la xarxa. Així, s'independitza completament la implementació de la seva disposició dels components.

3.1 MÒDULS SIMPLES

Els mòduls simples són els components actius del model i es defineixen amb la paraula **`simple`**.

```
simple ModulSimple {
    parameters:
    gates:
}
```

3.1.1 PARAMETERS

Els paràmetres es poden considerar com les variables que pertanyen al mòdul. S'especifiquen segons el patró *`tipus nom_variable`* (on el nom de la variable, per convenció, es comença amb minúscula). Els tipus poden ser `double`, `int`, `bool`, `string` i `xml`, i es poden declarar com a `volatile` (de manera que s'avalua l'expressió corresponent cada vegada que cal utilitzar el paràmetre). Per als tipus numèrics, també es pot especificar la unitat de mesura. Els paràmetres es poden inicialitzar o no. Si s'inicialitzen, després no es poden inicialitzar novament al fitxer INI. Dins els paràmetres també es poden especificar propietats del mòdul com, per exemple, la icona que es vol utilitzar per representar el mòdul. Les propietats s'identifiquen amb una @.

```
simple ModulSimple {
    parameters:
        int id = 1;
        int capacity;
        volatile double interarrivalTime @unit(s) = default(exponential(1s));
        @display("i=block/source");
    gates:
}
```

3.1.2 GATES

Els accessos (**gates**) constitueixen els punts d'entrada i/o sortida del mòdul per on es reben i/o s'envien els missatges des de o cap a un altre mòdul. Es poden declarar com a punts d'entrada (`input`), de sortida (`output`) o bidireccionals (`inout`). A més, es poden crear vectors de **gates** simplement afegint claus ('[ ]') després del nom.

```
simple ModulSimple {
    parameters:
        int id = 1;
        int capacity;
        volatile double interarrivalTime @unit(s) = default(exponential(1s));
        @display("i=block/source");
    gates:
        input in;
        output out[];
}
```

Tant la secció **parameters** com la secció **gates** són opcionals.

3.2 MÒDULS COMPOSTOS

Un mòdul compost agrupa altres mòduls (simples o compostos) en una unitat major. La paraula clau per definir un mòdul compost és **module** i, igual que un mòdul simple, pot tenir **parameters** i **gates**, a més d'altres seccions (totes opcionals).

```
module Host {
   parameters:
   gates:
   submodules:
   connections:
}
```

3.2.1 SUBMODULES

A la secció de **submodules** s'especifiquen els mòduls (simples o compostos) que conformen el mòdul. Cada submòdul, a més, pot definir paràmetres, fixar la longitud dels vectors d'accessos i redefinir propietats.

```
module Node {
    gates:
        inout port[];
    submodules:
        routing: Routing {
            parameters:
                routingTable = "routingtable.txt";
            gates:
                in[sizeof(port)];
                out[sizeof(port)];
        }
        queue[sizeof(port)]: Queue {
            @display("t=queue id $id");
            id = 1000+index;
        }
    connections:
}
```

3.2.2 CONNECTIONS

A la secció de **connections** s'especifiquen les connexions entre els accessos dels diferents mòduls utilitzant fletxes. Entre les fletxes es poden especificar els paràmetres i les propietats que es vol que tingui el canal.

```
module NodeGroup {
    submodules:
        node1: Node;
        node2: Node;
    connections:
        node1.port++ <--> {datarate = 100Mbps} <--> node2.port++;
}
```

3.2.3 CHANNELS

Els **channels** encapsulen paràmetres i aspectes relacionats amb les connexions. Es poden considerar mòduls simple en el sentit que tenen classes C++ al darrera.

```
channel C extends ned.DatarateChannel {
    datarate = 100Mbps;
}

module NodeGroup {
    submodules:
        node1: Node;
        node2: Node;
    connections:
        node1.port++ <--> C <--> node2.port++;
}
```

3.3 NETWORKS

Les xarxes es defineixen amb la paraula clau **network** i són mòduls (normalment compostos) sense **gates**.

```
network Network {
    @display("i=block/network");
    submodules:
        queue: Queue;
        sink: Sink;
        source: Source;
    connections:
        source.out --> queue.in++;
        queue.out --> sink.in++;
}
```

4 MÒDULS

4.1 CONCEPTES DE SIMULACIÓ

Un sistema d'esdeveniments discrets és un sistema on els canvis d'estat (esdeveniments) succeeixen en instants de temps determinats i s'assumeix que no succeeix res (no hi ha canvi d'estat) entre dos esdeveniments consecutius. Els sistemes que es poden veure com a sistemes d'esdeveniments discrets es poden modelar utilitzant simulació d'esdeveniments discrets (DES). Les xarxes d'ordinadors s'acostumen a veure com a sistemes d'esdeveniments discrets, on dos dels esdeveniments són l'inici i el final de la transmissió d'un paquet.

L'instant en el que ocorre un esdeveniment s'acostuma a anomenar **timestamp**. A OMNeT++ s'anomena **arrival time** perquè la paraula **timestamp** està reservada per a un altre ús. El temps dins el model s'anomena **temps de simulació**, **temps de model** o **temps virtual** (per diferenciar-lo del **temps real** o del **temps de CPU** que fan referència al temps total que ha durat la simulació o al consum de temps de CPU que ha requerit, respectivament).

Els simuladors d'esdeveniments discrets acostumen a mantenir el conjunt d'esdeveniments futurs en una estructura de dades anomenada FES (Future Event Set) i el funcionament segueix el pseudocodi següent:

```
initialize - (fase que inclou construir el model i inserir els esdeveniments inicials al FES)
while (FES no és buit i la simulació no ha acabat) {
       treure primer element del FES
       t:= timestamp de l'esdeveniment extret del FES
       processat de l'esdeveniment (que pot suposar incloure esdeveniments al FES o eliminar-ne)
}
finish simulation - (fase que inclou escriure estadístiques, resultats, etc.)
```

Els simuladors d'esdeveniments discrets acostumen a mantenir el conjunt d'esdeveniments futurs en una estructura de dades anomenada FES (Future Event Set) i el funcionament segueix el pseudocodi següent:

OMNeT++ utilitza missatges per representar esdeveniments. Cada esdeveniment és representat per una instància de la classe cMessage o una de les seves subclasses. Els missatges s'envien d'un mòdul a un altre, de manera que

l'esdeveniment succeeix al mòdul on s'envia el missatge i el temps d'execució és el temps d'arribada del missatge. Els esdeveniments que suposen l'expiració d'un temporitzador s'implementen fent que u n mòdul s'enviï un missatge a sí mateix.

Els esdeveniments s'extreuen del FES en funció del **temps d'arribada** (per mantenir la causalitat). Si dos esdeveniments tenen el mateix temps d'arribada, s'executa primer el de menor prioritat de programació (un paràmetre que l'usuari pot fixar). Si coincideix la prioritat de programació, l'esdeveniment que s'ha introduït al FES abans, és el que s'executa primer.

A OMNeT++, el temps de simulació es pot obtenir amb la funció `simTime()` que retorna un tipus C++ anomenat `simtime_t` . El FES està implementat utilitzant una **heap binària** (l'estructura de dades més utilitzada per a aquest propòsit).

4.2 CONCEPTES BÀSICS DELS MÒDULS SIMPLES

Els models de simulació d'OMNeT++ està compostos de mòduls i connexions. Els mòduls poden ser simples (atòmics) o compostos. Els mòduls simples són els components actius del model i el seu comportament el defineix l'usuari com codi C++. Les connexions poden tenir canals associats. Els canals encapsulen el comportament del canal (temps de propagació i transmissió, taxa d'error i possiblement d'altres) i també es poden programar en C++.

Els mòduls i els canals són components. Així doncs, les classes abstractes cModule i cChannel són ambdues subclasses de la classe cComponent. La classe cModule té dues subclasses: cSimpleModule i cCompoundModule, i l'usuari defineix mòduls simples fent subclasses de la classe cSimpleModule. Anàlogament, l'usuari pot crear nous canals fent subclasses de la classe cChannel o dels tres tipus de canal pre-definits: cIdealChannel, cDelayChannel i cDatarateChannel.

Per definir un mòdul simple cal registrar la classe a OMNeT++ (utilitzant la macro `Define_Module(nom_modul)` al fitxer .cc corresponent) i especificar el comportament del mòdul re-definint els mètodes següents:

- `void initialize()`. Aquesta funció s'invoca després que OMNeT++ hagi creat la xarxa (és a dir, després de crear els mòduls i connectar-los segons les definicions) i és el lloc adient per inicialitzar les variables que defineixen el comportament del mòdul.

- `void handleMessage(cMessage *msg)`. Aquesta funció s'invoca quan el mòdul rep un missatge. La funció ha de processar el missatge i retornar immediatament després. Hi ha tres funcions relacionades amb el seu ús:
    - `send()`. Per enviar un missatge a un altre mòdul.
    - `scheduleAt()`. Per programar un esdeveniment pel propi mòdul.
    - `cancelEvent()`. Per esborrar un esdeveniment programat amb `scheduleAt()`.

- `void activity()`. Aquesta funció és similar a `handleMessage` però operant en mode **threat**.

- `void finish()`. Aquesta funció s'invoca quan la simulació finalitza correctament i és el lloc adequat per realitzar el càlcul i el bolcat d'estadístiques.

5 MISSATGES

Els missatges són un element fonamental dels models d'OMNeT++ ja que representen els esdeveniments, els paquets, les comanes i, en general, qualsevol tipus d'entitat del domini del model de simulació. Els missatges estan representats per la classe cMessage i la seva subclasse cPacket. La subclasse cPacket s'utilitza per representar els paquets (trames, datagrames, paquets de transport, etc.) d'una xarxa de comunicacions. La classe cMessage s'utilitza per tota la resta de coses.

A cada missatge se li pot assignar *name* (característica molt útil per identificar-lo quan s'executa la simulació utilitzant la interfície gràfica), *type*, *schedulingPriority* i *timpestamp*.

Entre les funcions més importants de la classe cMessage hi ha la funció `dup()`, que permet clonar, la funció `isSelfMessage()` que permet determinar si el paquet l'ha generat el propi mòdul, i la funció `isScheduled()` que permet saber si un missatge està programat o no. La llista completa de mètodes de la classe cMessage (així com la de totes les classes que defineix OMNeT++) es pot consultar a: http://www.omnetpp.org/doc/omnetpp/api/

Per definir un missatge amb els camps que desitja un usuari cal generar una subclasse de cMessage. Per fer-ho, es pot crear un fitxer de declaració de missatge (`.msg`) i utilitzar l'eina *opp_msgc* (ja integrada a l'IDE) per traduir-lo a una classe C++. Així, a partir de la definició del fitxer amb extensió `.msg` es creen automàticament els fitxers `.h` i `.cc` amb els mètodes `get` i `set` corresponents per a la lectura i escriptura dels nous atributs definits pel missatge.

```
message ElMeuMissatge {
        int nouCamp;
}
```

La classe cPacket amplia la classe CMessage amb camps útils per representar paquets com, per exemple, el camp de longitud de paquet. Igual que es fa amb la classe CMessage, es poden generar subclasses de la classe cPacket utilitzant fitxers `.msg` (però, en aquest cas, declarant l'element com a **packet** enlloc de fer-ho com a **message**.

```
packet ElMeuPacket {
        int nouCamp;
}
```

6 FITXERS INI

Els fitxers INI contenen els paràmetres de configuració dels models a simular.

6.1 SECCIONS DEL FITXERS DE CONFIGURACIÓ

Al fitxer de configuració és indispensable que hi figuri la xarxa a simular. En el cas més simple, la xarxa i tots els paràmetres es configuren a la secció `[General]` del fitxer `*.ini`. No obstant, és possible definir múltiples configuracions dins un mateix fitxer utilitzant la paraula clau `Config` seguida del nom de la secció entre claus: `[Config nom_seccio]`. Els paràmetres que es configuren a la secció general, tenen efecte sobre totes les seccions. És a dir, aquells paràmetres que es volen utilitzar en totes les simulacions es poden agrupar en la secció general, deixant els paràmetres particulars de cada simulació per a les diferents seccions. A més a més d'especificar la xarxa que es vol simular i els paràmetres relacionats amb la configuració dels mòduls, al fitxer de configuració també es poden establir els valors d'alguns paràmetres de la simulació com per exemple:

- `sim-time-limit = valor_temporal`. Indica la durada màxim de la simulació en temps de temps d'observació.
- `output-vector-file = nom_fitxer.vec`. Especifica el nom del fitxer on es guardaran les dades del mostreig realitzat durant la simulació.
- `output-scalar-file = nom_fitxer.sca`. Especifica el nom del fitxer on es guardaran les dades estadístiques resultants de la simulació.
- `cmdenv-express-mode = true`. Habilita el mode ràpid de simulació en la interfície **Cmdenv**. Les simulacions es poden executar utilitzant la interfície **Tkenv** o la **Cmdenv**. Per defecte, es selecciona la interfície **Tkenv** que proporciona un entorn gràfic on poder observar el model a simular.
- `repeat = num_repeticions`. Determina el número d'execucions amb uns mateixos paràmetres d'entrada que es desitja realitzar (molt utilitzat quan es volen realitzar diferents execucions amb diferents llavors del generadors de variables aleatòries).
- `warmup-period = valor_temporal`. Estableix la durada de l'interval transitori.

7 CONCLUSIONS

Per completar aquest document introductori a l'OMNeT++ caldria parlar de la presa d'estadístiques, el compilat i linkat del simulador, i l'anàlisi dels resultats. Aquests aspectes es tracten de forma experimental al tutorial TicToc que es proporciona l'OMNeT++ (http://www.omnetpp.org/doc/omnetpp/tictoc-tutorial/) i que es recomana realitzar per familiaritzar-se amb l'entorn de treball.